Canasta Manager

Aufgaben

Das Kartenspiel Canasta kann zu zweit oder zu dritt, aber auch mit zwei bis drei Teams von je zwei Personen gespielt werden. Ein Hobby-Kartenspieler versucht, den Ablauf eines Canasta-Spiels digital zu modellieren und zu implementieren.

1 Objektorientierte Modellierung und Implementierung eines Canasta-Spiels
Ein UML-Klassendiagramm finden Sie in Material 1. Die nachfolgenden Aufgaben beziehen sich auf ein Canasta-Spiel für zwei Personen. Hauptziel jeder Spielerin beziehungsweise jedes Spielers (im Folgenden Spieler genannt) ist es, jeweils sieben Karten gleicher Art (einen „Canasta“) zu sammeln und auszulegen, also zum Beispiel sieben Damen oder sieben Achten. Für die Lösung der Aufgaben sind keine Kenntnisse der Spielregeln erforderlich.

1.1 In einer Spielsituation hat der Spieler Christian Müller noch vier Karten auf der Hand: eine Herz Dame, eine Karo Dame, einen Joker und eine Kreuz Sieben. Entwickeln und zeichnen Sie ein Objektdiagramm für den Spieler und seine Karten.

Hinweis: Verwenden Sie neben dem Klassendiagramm in Material 1 auch die Angaben zu den Karten in Material 3.

(6 BE)

1.2 Zu Beginn jedes Spiels werden die Karten des verwendeten Kartensatzes gemischt. Implementieren Sie eine Methode `mischen()` der Klasse `CanastaManager`.

Hinweis: Die Dokumentation der Klasse `Random` finden Sie in Material 2.

(4 BE)

1.3 Die Methode `ausfuehrenSpielzug()` der Klasse `CanastaManager` wird abwechselnd für jeden Spieler durchgeführt. Sie zieht die oberste Karte vom Haufen (1. Position im Container `haufen`) und fügt sie sortiert in die Handkarten des Spielers ein. Dann wählt der Spieler eine Karte auf seiner Hand aus, die er ablegen möchte. Dazu wählt er den Index dieser Karte und entfernt sie somit von seiner Hand. Anschließend wird diese Karte als oberste auf den Stapel (Container `stapel`) abgelegt. Entwickeln und zeichnen Sie ein Sequenzdiagramm für die Methode `ausfuehrenSpielzug(spieler)`.

Hinweise: Verwenden Sie die Vorlage in Material 4, die Hinweise in Material 1 sowie die Klassendokumentation der Klasse `List` in Material 2.

(8 BE)

1.4 Überführen Sie die Klassen `Spieler` und `Karte` in Anweisungen einer objektorientierten Programmiersprache. Implementieren Sie die Konstruktoren und die aufgeführten Methoden mit Ausnahme der Methode `waehleKarte()` der Klasse `Spieler`.

Hinweise: Die Dokumentation der Klasse `List` finden Sie in Material 2 und das Datenblatt für die Karten sowie Hinweise zu ihrer Sortierung in Material 3. Die Hinweise zum Klassendiagramm in Material 1 sind zu beachten.

(14 BE)

1.5 In Material 3 finden Sie ein Beispiel für eine mit Karten belegte Auslegefläche und die Regeln für die Berechnung der Punkte. Dazu wurden den Elementen des zugehörigen zweidimensionalen Arrays (`flaeche[ZEILEN][MAX_ANZAHL]`) die entsprechenden Spielkarten zugewiesen. Die hierfür erzielte Punktezahl soll in der Methode `ermittlePunkte()` der Klasse `Auslegeflaeche` bestimmt werden.
Implementieren Sie die Methoden `ermittlePunkte()`, `punkteZeile()` und `punkteCanasta()`.

(8 BE)

1.6 Zum Sortieren der Handkarten des Spielers soll ein Quicksort-Algorithmus zum Einsatz kommen. Zum Testen wurde in einer Klasse `QuicksortArray` ein Algorithmus für die Sortierung ganzer Zahlen in Arrays implementiert. Zugehörige statische Methoden sind in Material 5 in drei Struktogrammen dargestellt. Die Hilfsmethode `tauscheElemente`() wurde verwendet.

1.6.1 Erklären Sie die Vorgehensweise des vorliegenden Sortieralgorithmus.

Hinweis: Die Vorgehensweise in der Hilfsmethode `teileArray()` soll nicht in die Erklärung einbezogen werden.

(5 BE)

1.6.2 Um nachzuvollziehen, wie eine fremde Software funktioniert, kann sie zunächst auf dem Papier an einem kleinen Beispiel getestet werden („Schreibtischtest"). Bestimmen Sie auf Grundlage des vorliegenden Sortieralgorithmus die ersten beiden Ausgabewerte von `q`. Geben Sie dabei nach jeder Änderung im Array immer das gesamte aktuelle Array an.

Hinweise: Gehen Sie schrittweise die Struktogramme, beginnend bei der `main()`-Methode, durch. Geben Sie bei möglichen Methodenaufrufen, Bedingungen oder Schleifen immer die konkreten Parameterwerte an. Beispielhaft sind in Material 6 bereits die ersten Methodenaufrufe und Ausgabewerte angegeben.

(8 BE)

1.6.3 In Analogie zum vorliegenden Algorithmus soll später in einer Klasse `QuicksortKarten` (siehe Material 7) eine Methode `qsort()` entwickelt werden, mit deren Verwendung in der Klasse `Spieler` Handkarten sortiert werden können. Implementieren Sie die zu `teileArray()` in Material 5 entsprechende statische Methode `teileKarten()` sowie die statische Hilfsmethode `tauscheKarte()`, die die Listenelemente mit den Indices a und b vertauscht und anschließend die Liste zurückgibt.

Hinweis: Die Klasse `Karte` implementiert das Interface `Comparable<Karte>` (siehe Material 2).

(7 BE)

2 Zudem wird für professionelle Canasta-Turniere eine Datenbank für die Verwaltung von zurückliegenden und zukünftigen Canasta-Spielrunden entwickelt.
Ein erstes Entity-Relationship-Modell (ERM) für das Canastaspielen in Zweierteams ist in Material 8 dargestellt. Ein Canasta-Spiel besteht aus mehreren Spielrunden, deren Punkte so lange summiert werden, bis ein Team mindestens 5000 Punkte erreicht hat und somit als Gewinner feststeht.
Hinweis: Es ist davon auszugehen, dass nur ein Team mehr als 5000 Punkte erreicht.

2.1 Beschreiben Sie die Grundbausteine des vorliegenden Entity-Relationship-Modells unter Einbeziehung von Beispielen. Geben Sie den Verwendungszweck eines ERM an.

(4 BE)

2.2 Überführen Sie das ERM (Material 8) in das Relationenmodell in der 3. Normalform. Begründen Sie ihre Entscheidungen.

Hinweis: Alle Relationen sind in der Schreibweise `Relation(PK, Attribut, ..., FK#)` anzugeben.

(8 BE)

2.3 Einige Daten der Datenbank sollen ergänzt, geändert bzw. ausgewertet werden. Die Attribute `cSpielID` und `teamID` werden automatisch vom System vergeben.

2.3.1 Frau Petra Schmidt, geboren am 19. Juli 2001, möchte erstmalig an den Canastaspielen teilnehmen, und zwar zusammen im Team mit der Stammspielerin Astrid Treusch, und zunächst ohne Wertung. Frau Schmidt wurde mit der Spielernummer 57 in die Datenbank eingetragen. Entwickeln Sie die SQL-Anweisungen, um die Spielerin und die Teamzusammenstellung anzulegen.

Hinweis: Es gibt nur eine Spielerin mit dem Namen Astrid Treusch.

(5 BE)

2.3.2 Es wurde ein neuer analoger Kartensatz angeschafft und mit der ID 13 in die Datenbank aufgenommen. Dieser Kartensatz soll an allen zukünftigen Spieltagen den nicht mehr vollständigen Kartensatz mit der ID 1 ersetzen, welcher aus der Datenbank zu entfernen ist. Entwickeln Sie die SQL-Anweisungen zur Aktualisierung der Datenbank unter Berücksichtigung der referentiellen Integrität.

(5 BE)

2.3.3 Für eine Statistik sollen alle gewonnenen Spiele ausgegeben werden, unter Angabe der teamID des Gewinnerteams, der spielID und der erreichten Punktzahl. Implementieren Sie dafür eine SQL-Anweisung.

(2 BE)

2.3.4 Entwickeln Sie eine SQL-Anweisung, die ermittelt, welche Kartensätze in Canastaspielen jeweils über 100-mal eingesetzt worden sind. Die Anzahl, wie oft die betreffenden Kartensätze verwendet worden sind, ist dabei absteigend sortiert auszugeben.

(5 BE)

2.3.5 Für jeden Kartenspieler, unter Angabe seines vollständigen Namens und der Spielernummer, ist die Gesamtpunktzahl über alle Canastaspiele gesucht, an denen er beteiligt war, vorausgesetzt, sein Team war mit dem Bewertungsmodus `true` gemeldet.
Entwickeln Sie für diese Auswertung eine entsprechende SQL-Anweisung,

(4 BE)

2.4 Die Datenbank soll zur Verwaltung von weiteren beliebten Kartenspielen, zum Beispiel Skat oder Doppelkopf, erweitert werden. Außerdem sollen klassische Brettspiele dazukommen, zum Beispiel Schach oder „Mensch ärgere dich nicht“. Folgende Festlegungen sind zu berücksichtigen:

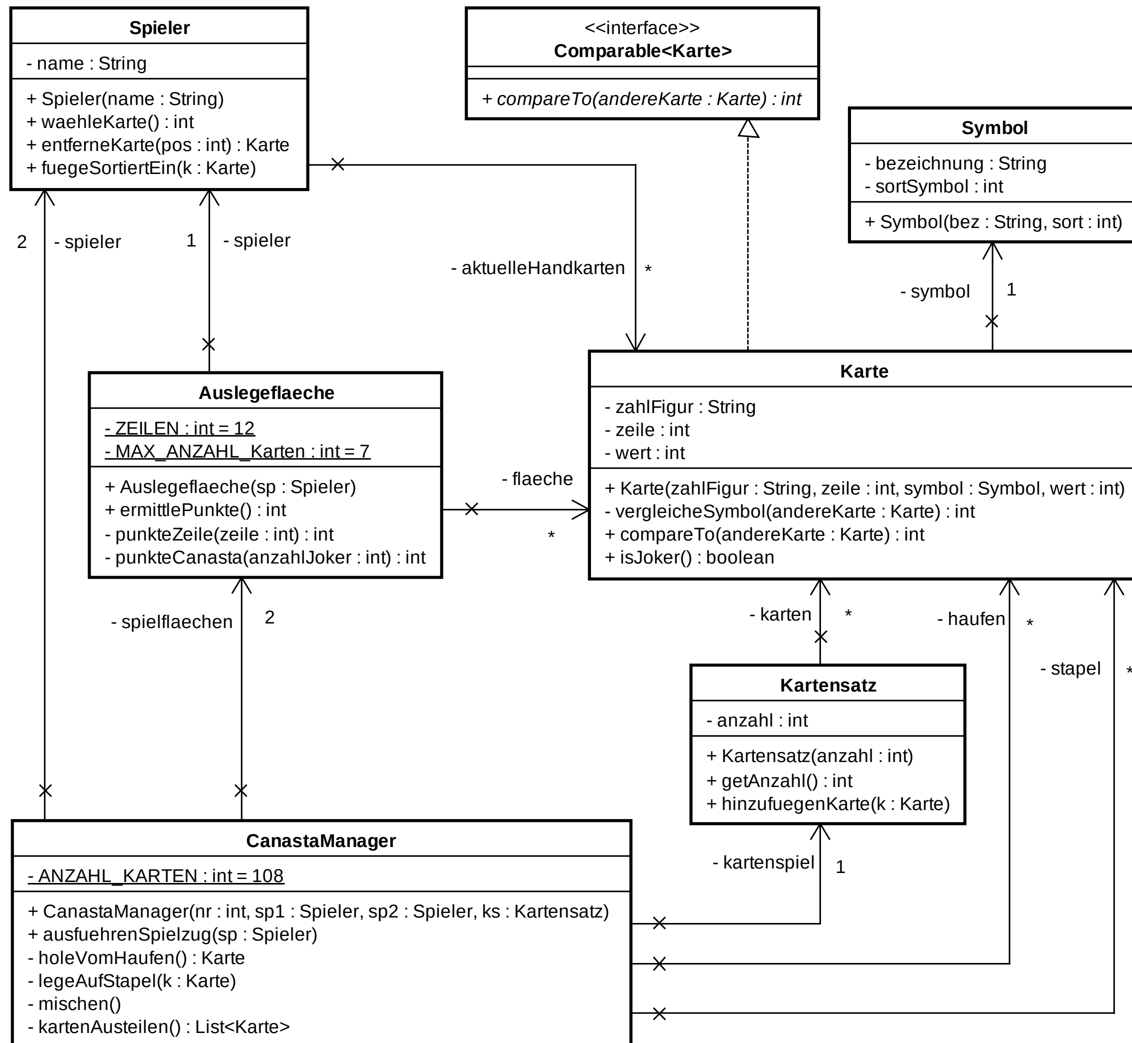
– Für jedes Gesellschaftsspiel können Spiele durchgeführt werden.
– Für ein Gesellschaftsspiel wird neben seiner Bezeichnung die minimal und die maximal benötigte Personenzahl von Spielern sowie ein Hyperlink zu den Spielregeln gespeichert.
– Bei Kartenspielen wird die Anzahl der Karten und bei den Brettspielen die Anzahl der Würfel sowie die Anzahl der Spielsteine gespeichert.
– Jede Spieldurchführung kann über eine Turnierverwaltung verfügen, und jede Turnierverwaltung bezieht sich auf genau eine Spieldurchführung.
– Für die Turnierverwaltung soll die Bezeichnung des Turniers vermerkt sein. Außerdem wird der Modus gespeichert, zum Beispiel „jeder gegen jeden“ oder „in zwei Gruppen parallel“.

Entwickeln und zeichnen Sie die hier geforderten Erweiterungen des in Material 9 vorliegenden ERM.

(7 BE)

Material 1

UML-Klassendiagramm Canasta-Spiel



Hinweise:
Klasse `Spieler`:

- `waehleKarte()` gibt die Position der vom Spieler gewählten Handkarte zurück.
- `entferneKarte(pos: int)` entfernt die entsprechende Karte mit der übergebenen Position aus der Hand des Spielers und gibt sie zurück.
- `fuegeSortiertEin()` fügt die übergebene Karte, entsprechend der `compareTo()`-Methode des Interfaces `Comparable` (Material 2) sortiert in die Handkarten des Spielers ein.

Klasse `Karte`:

- `vergleicheSymbol()` liefert den Wert 0, wenn beide Karten den gleichen Wert von `sortSymbol` (Klasse `Symbol`) haben, -1, wenn das Karten-Objekt davor liegt, ansonsten 1. Die Sortierung nach `sortSymbol` stellt bei der Sortierung mit `compareTo()` (siehe Interface `Comparable` Material 2) das untergeordnete Sortierkriterium dar. Es wird zuerst nach der Zeile und dann nach `startSymbol` sortiert.
- `isJoker()` gibt `true` zurück, wenn die Karte ein Joker oder eine 2 ist, ansonsten `false`.

Auf alle Attribute kann mittels get- und set-Methoden zugegriffen werden.

Material 2

Klassendokumentationen

Interface `Comparable<T>`

`compareTo(o: T): int`
Liefert einen int-Wert
< 0 wenn das Objekt vor dem Parameter `o` liegt
= 0 wenn das Objekt gleich `o` ist
> 0 wenn das Objekt nach dem Parameter `o` liegt

Klasse `List`

List<T>
+ List<T>() + add(obj : T) + add(index : int, obj : T) + contains(obj : T) : boolean + get(index : int) : T + set(index : int, obj : T) : T + remove(index : int) : T + remove(obj : T) : boolean + size() : int

`List<T>()`
erzeugt eine generische Liste mit Elementen des Typs `T`.

`add(obj: T)`
hängt das Objekt `obj` vom Typ `T` am Ende der Liste an.

`add(index: int, obj: T)`
fügt das Objekt `obj` vom Typ `T` an der Position `index` in die Liste ein.

`contains(obj: T): boolean`
liefert `true`, wenn das Objekt `obj` in der Liste enthalten ist, ansonsten `false`.

`get(index: int): T`
liefert das Listenelement an der Position `index` zurück bzw. `null`, falls `index` negativ oder größer gleich der Anzahl der momentan enthaltenen Elemente ist.

`set(index: int, obj: T): T`
ersetzt das Listenelement an der Position `index` durch das Objekt `obj` vom Typ `T` und gibt es zurück.

`remove(index: int): T`
entfernt das Objekt vom Typ `T` an der Position `index` aus der Liste und gibt es zurück.

`remove(obj: T): boolean`
entfernt das Objekt `obj` aus der Liste. Falls `obj` mehrmals in der Liste enthalten ist, wird nur das erste Vorkommen entfernt. Der Rückgabewert ist `true`, falls das Objekt gefunden und entfernt wurde, sonst `false`.

`size(): int`
liefert die Anzahl der Elemente in der Liste zurück.

Klasse `Random`

Random
+ Random() + nextInt() : int + nextInt(n : int) : int

`Random()`
erzeugt ein Objekt der Klasse Random

`nextInt(): int`
erzeugt eine Zufallszahl aus dem Bereich der positiven Zahlen.

`nextInt(n: int): int`
erzeugt eine Zufallszahl aus dem Bereich 0 (einschließlich) bis n (ausschließlich) der ganzen Zahlen.

Material 3

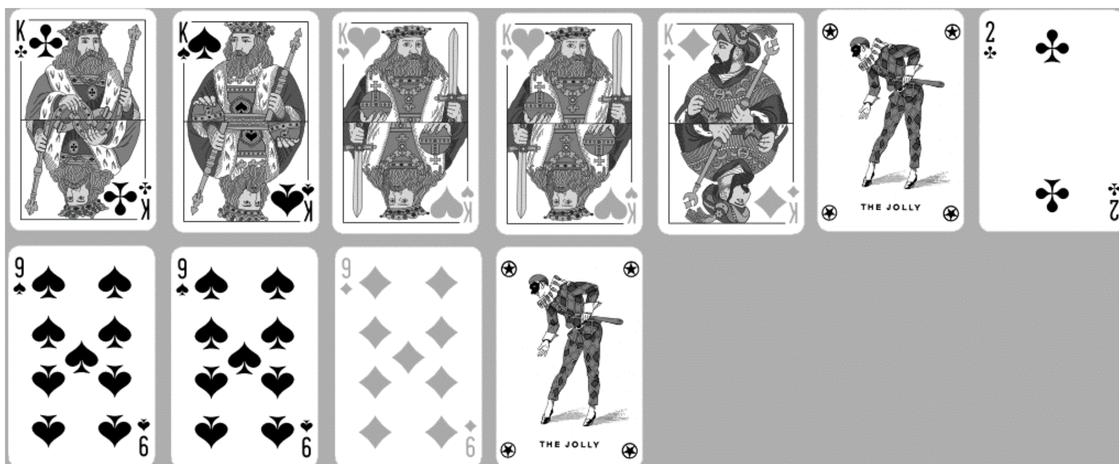
Sortierung und Wertung der Spielkarten

zahlFigur	zeile	symbole	wert
Bube	0	Kreuz, Pik, Herz, Karo	10
Dame	1	Kreuz, Pik, Herz, Karo	10
König	2	Kreuz, Pik, Herz, Karo	10
Ass	3	Kreuz, Pik, Herz, Karo	20
4	4	Kreuz, Pik, Herz, Karo	5
5	5	Kreuz, Pik, Herz, Karo	5
6	6	Kreuz, Pik, Herz, Karo	5
7	7	Kreuz, Pik, Herz, Karo	5
8	8	Kreuz, Pik, Herz, Karo	5
9	9	Kreuz, Pik, Herz, Karo	5
10	10	Kreuz, Pik, Herz, Karo	10
Joker	11	Joker	50
2 (Einsatz als Joker)	11	Kreuz, Pik, Herz, Karo	20

bezeichnung	sortSymbol
Kreuz	4
Pik	3
Herz	2
Karo	1
Joker	0

Hinweise zur Sortierung der Karten: In der Hand des Spielers werden die Karten nach ihrem Attribut `zeile` sortiert, zum Beispiel alle Asse (zum Beispiel Kreuz-Ass und Herz-Ass) nebeneinander gesteckt, zu erkennen am gleichen Wert 3 ihres Attributs `zeile`. Darüber hinaus ist für eine eindeutige Sortierung der Karten auf der Hand des Spielers die Sortierung nach der Bezeichnung der Karte, etwa ob diese Karte ein Kreuz- oder ein Herz-Ass ist, von Bedeutung. Eine Kreuz-Karte hat den Wert 4 des Attributs `sortSymbol`, und eine Herz-Karte den Wert 2. Joker können sich in allen Zeilen befinden, sofern es kein Canasta aus Jokern ist. Die Sonderregel der Karte mit der Zahl 3 wird nicht betrachtet.

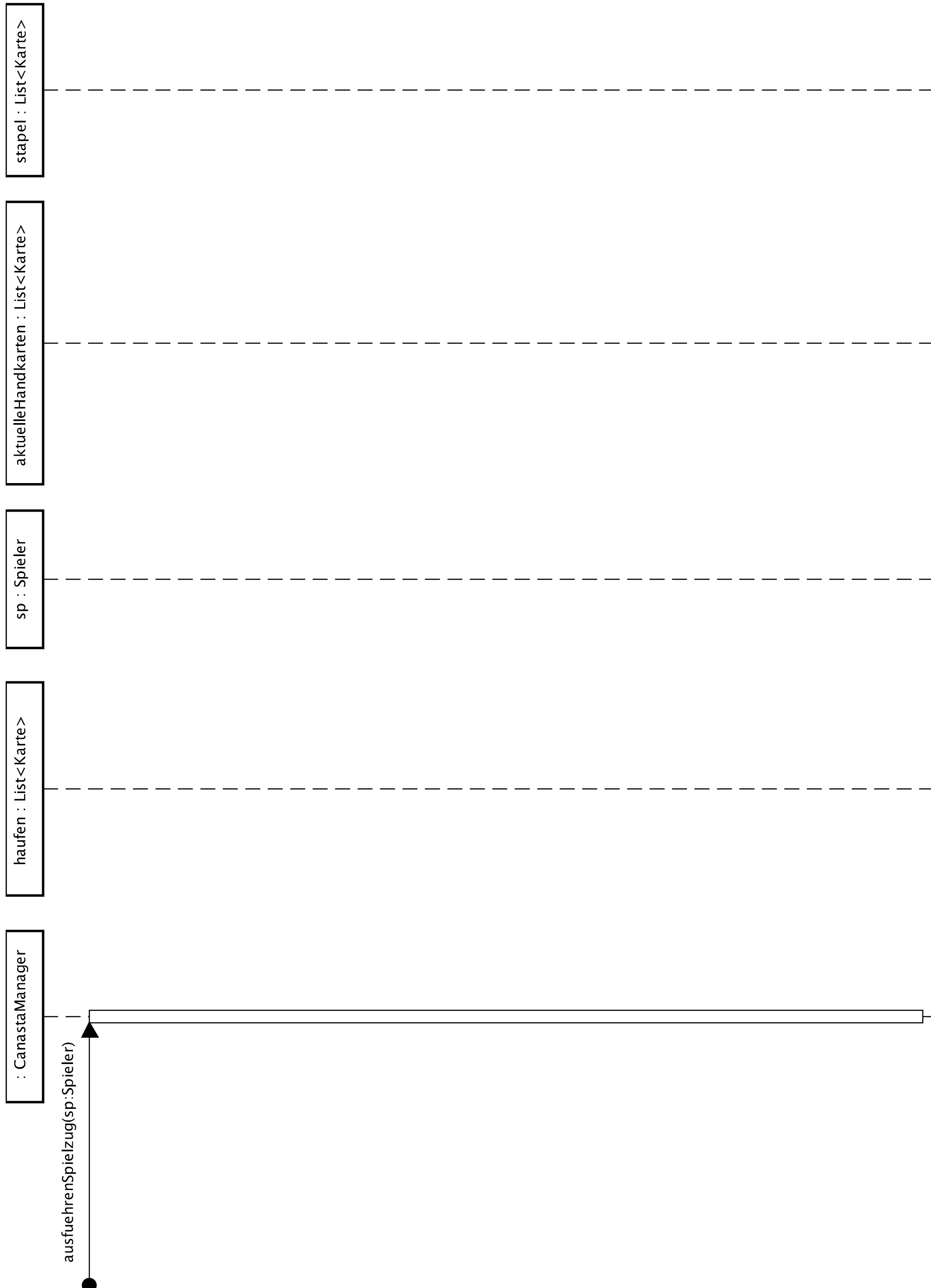
Auslegefläche Beispiel



Hinweise zur Auslegefläche: Von den Zeilen 0 bis 11 einer Auslegefläche sind die Kartenreihen für `zeile=2` und `zeile=9` dargestellt. Zur Berechnung der Gesamtpunkte werden die Werte der Karten addiert. Zusatzpunkte gibt es für jeden Canasta: 1000 für einen Canasta bestehend aus Jokern, 500 für einen Canasta ohne Joker und ansonsten 300 Punkte. Beispiel: Die Punktezahlen der Karten für diese Reihen betragen 120 und 65. Dazu kommt die Wertung für einen Canasta mit Jokern von 300 Punkten. Daraus entsteht eine Gesamtpunktezahl von 485.

Material 4

Vorlage UML Sequenzdiagramm



Material 5

Struktogramme einer Klasse QuicksortArray

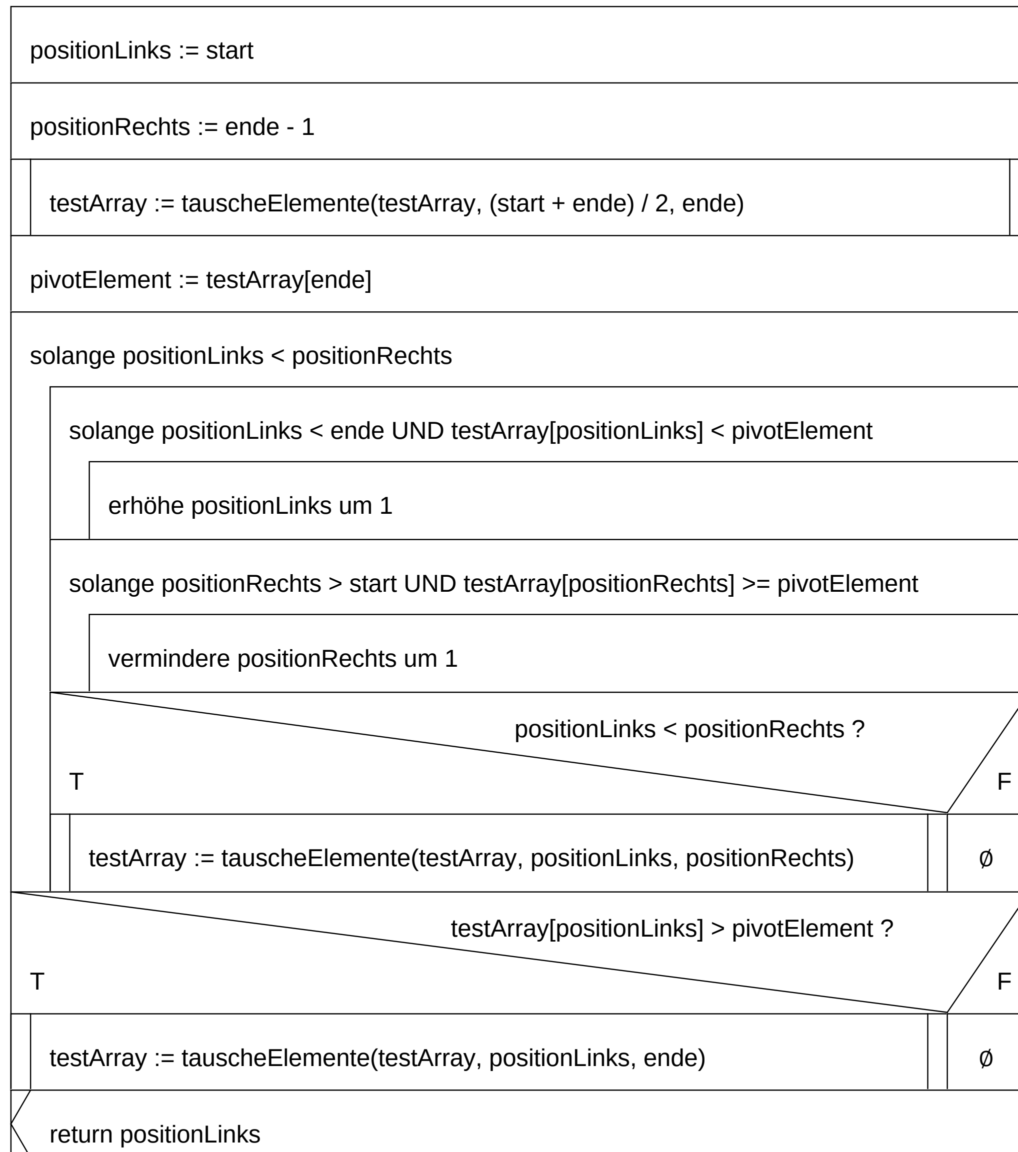
main()

```
testArray := {21, 16, 7, 14}
| ergebnisArray := quicksort(testArray, 0, testArray.length - 1) |
| ausgebenArray(ergebnisArray) |
```

quicksort(testArray: int[]; start: int; ende: int): int[]

```
start < ende ?
T:
  | q := teileArray(testArray, start, ende) |
  Ausgabe: q
  | testArray := quicksort(testArray, start, q - 1) |
  | testArray := quicksort(testArray, q + 1, ende) |
F:
  ø
return testArray
```

Fortsetzung Material 5



Material 6

Schreibtischtest QuicksortArray (Ausschnitt)

testArray:

0	1	2	3
21	16	7	14

QuicksortArray.quicksort(testArray,0,3)
start=0, ende=3, 0<3 ? ja

q=teileArray(testArray,0,3)
  positionLinks=0, **positionRechts=2**, m=1
testArray=tauscheElemente(testArray,1,3):

0	1	2	3
21	14	7	16

pivotElement=16
solange positionLinks<positionRechts: 0<2? ja
  solange positionLinks<ende: 0<3? ja und 21<16? nein
  solange positionRechts>start: 2>0? ja und 7>=16? nein
  positionLinks<positionRechts: 0<2? ja
  testArray=tauscheElemente(testArray,0,2):

0	1	2	3
7	14	21	16

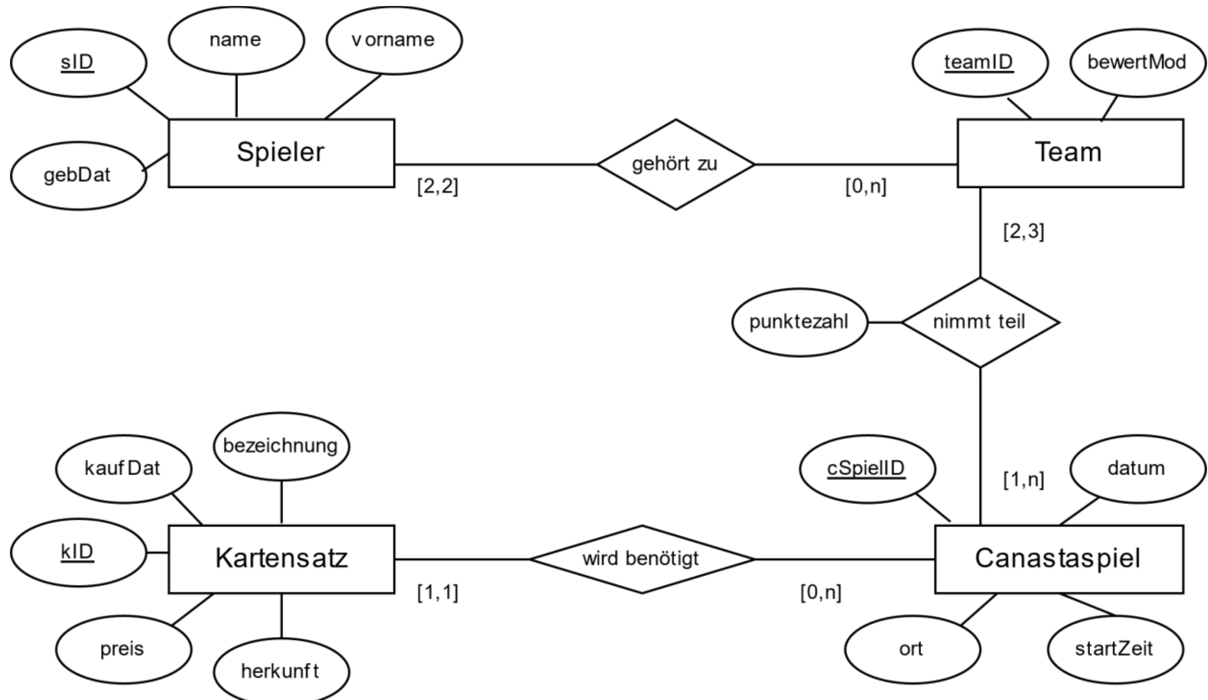
Material 7

Klasse QuicksortKarten

QuicksortKarten
+ qsort(karten : List<Karte>, links : int, rechts : int) : List<Karte> - teileKarten(karten : List<Karte>, start : int, ende : int) : int - tauscheKarte(karten : List<Karte>, a : int, b : int) : List<Karte>

Material 8

Entity-Relationship-Modell



Hinweise: Eine Person kann an einem Tag an mehreren Canasta-Spielen bis 5000 Punkte teilnehmen, die aber eine unterschiedliche Startzeit haben müssen. Das Attribut `bewertMod` ist `false`, wenn ohne Wertung gespielt wird, ansonsten `true`.

Material 9

Erweiterung ERM

